

# An XML Web strategy for scientific documents

Michel Goossens, CERN

24 May 2002

## 1 Introduction

Whereas the advent and the ready availability of the personal computer drastically reduced the production cost of electronic documents, the creation of the Web made distributing these documents worldwide a lot cheaper, easier, and faster. Taken together, these two developments have considerably changed the economic factors controlling the generation, maintenance, and dissemination of electronic documents. In addition, thanks to the development of the XML family of standards and the ubiquity of the platform-independent Java language, it is now possible to have a unified approach to the vast amount of information stored in databases and to handle their representation in various customizable forms.

$\text{\LaTeX}$  plays and will continue to play an important role in the integrated worldwide cyberspace, especially in the area of scientific documents, as well for text input and for rendering the output. In due course  $\text{\LaTeX}$  may be complemented by a semantically richer MathML representation. However,  $\text{\LaTeX}$ 's greatest impact will remain in the area of typesetting, with  $\text{\TeX}$  becoming an important intermediate format for generating high-quality printable PDF output. Various techniques are now becoming available to transform  $\text{\LaTeX}$  documents into PDF, HTML, or XML so that the information can be made available on the Web, and hence contribute to the richness and increasing wealth of the scientific hyper-culture of mankind. One does not have to choose between using  $\text{\LaTeX}$  or another markup technique; one can use whichever is more appropriate in a given situation.

Nowadays almost every company, school, organization, public or commercial utility, and, before long, every individual has or will want a presence on the Web. Also, or especially, in the scientific world most publications are made available on the Internet, and in many cases scientific results are available first on a Web site long before they are published in *paper* form in a recognized journal.

In the scientific world we have seen an extremely swift evolution from a kind of hesitation to embrace the new Web technology to the enthusiasm of using preprint databases to speed up the dissemination of information. The main problem that remains to be solved is *quality*, both in content and form. The quality of the content can be guaranteed by adopting the established peer review system, building upon the expertise of many of the existing publishing houses that find a new role as *information verification agents*. The quality of presentation is a problem that is not yet fully solved. There have been many public debates about whether current computer screens can provide the necessary detail to represent faithfully the visual multidimensional information inherent in a mathematical or chemical formula. Several attempts have already been made to come up with ways to overcome the coarseness of the computer screen (at best a few pixels per millimeter), keeping the flexibility of interactive hypertext searching possibilities.

Consider also the situation in some parts of the world, such as in Russia, and many countries in Asia, Africa and South America, that face severe financial constraints, and where it is often out of the question even to consider printing multiple copies of a highly technical document. Electronic dissemination via the Web is the only way, then, to publish. Thus the Web is not only an additional medium for the traditional publishing establishment, but a necessity for the larger part of the world to participate in sharing the information and benefit from the wealth and progress it creates.

### 3 Vavilov theory

Vavilov[5] derived a more accurate straggling distribution by introducing the kinematic limit on the maximum transferable energy in a single collision, rather than using  $E_{\max} = \infty$ . Now we can write[2]:

$$f(\epsilon, \delta s) = \frac{1}{\xi} \phi_v(\lambda_v, \kappa, \beta^2)$$

where

$$\begin{aligned}\phi_v(\lambda_v, \kappa, \beta^2) &= \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} \phi(s) e^{\lambda s} ds & c \geq 0 \\ \phi(s) &= \exp[\kappa(1 + \beta^2\gamma)] \exp[\psi(s)], \\ \psi(s) &= s \ln \kappa + (s + \beta^2\kappa) [\ln(s/\kappa) + E_1(s/\kappa)] - \kappa e^{-s/\kappa},\end{aligned}$$

and

$$\begin{aligned}E_1(z) &= \int_z^\infty t^{-1} e^{-t} dt & (\text{the exponential integral}) \\ \lambda_v &= \kappa \left[ \frac{\epsilon - \bar{\epsilon}}{\xi} - \gamma' - \beta^2 \right]\end{aligned}$$

The Vavilov parameters are simply related to the Landau parameter by  $\lambda_L = \lambda_v/\kappa - \ln \kappa$ . It can be shown that as  $\kappa \rightarrow 0$ , the distribution of the variable  $\lambda_L$  approaches that of Landau. For  $\kappa \leq 0.01$  the two distributions are already practically identical. Contrary to what many textbooks report, the Vavilov distribution *does not* approximate the Landau distribution for small  $\kappa$ , but rather the distribution of  $\lambda_L$  defined above tends to the distribution of the true  $\lambda$  from the Landau density function. Thus the routine `GVAVIV` samples the variable  $\lambda_L$  rather than  $\lambda_v$ . For  $\kappa \geq 10$  the Vavilov distribution tends to a Gaussian distribution (see next section).

Figure 1: Standard  $\LaTeX$  output in DVI viewer

## 2 The present situation

### 2.1 The traditional way $\TeX$ , DVI, and PDF

For almost two decades many scientists have used  $\LaTeX$  to source their documents and have relied on  $\TeX$  to typeset their results reliably, nicely and correctly. We illustrate this classical strategy by considering various representations of a compiled  $\LaTeX$  document. The standard  $\LaTeX$  DVI output is shown in Figure 1. The text, which is taken from a physics software manual, contains some simple inline and displayed math formulae.

With the advent of hypertext, the  $\TeX$  community also moved in that direction, and hypertext linking commands were introduced in DVI files. DVI viewers were extended to interpret these links and thus they were turned into simple hypertext browsers. In particular support for Sebastian Rahtz' `hyperref` package is now present in most DVI drivers. For instance, Figure 2 shows how `dviout` displays the example file on Microsoft Windows, highlighting the bibliographical citations in the first paragraph as links. The applications also support the loading of a Web browser when a URL link is encountered, but they cannot be integrated within a Web browser.

Typographic quality can also be guaranteed by generating a PDF file with `DVIPDFM` or `PDF $\TeX$`  (<http://www.tug.org/applications/pdftex/index.html>), or with `Acrobat Distiller` starting from a PostScript file. With a PDF browser, such as `Adobe Acrobat Reader`, `xpdf`, or `Ghostview` one can easily navigate through the document and exploit hypertext information present in the  $\LaTeX$  source ( $\LaTeX$  cross-references and bibliographic citations can be automat-

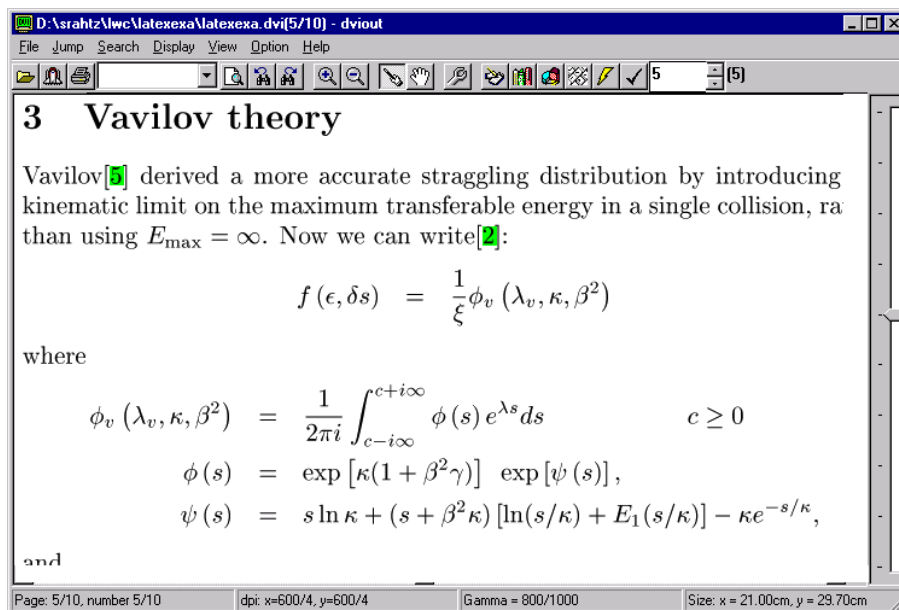


Figure 2: Hypertext DVI viewing with dviout

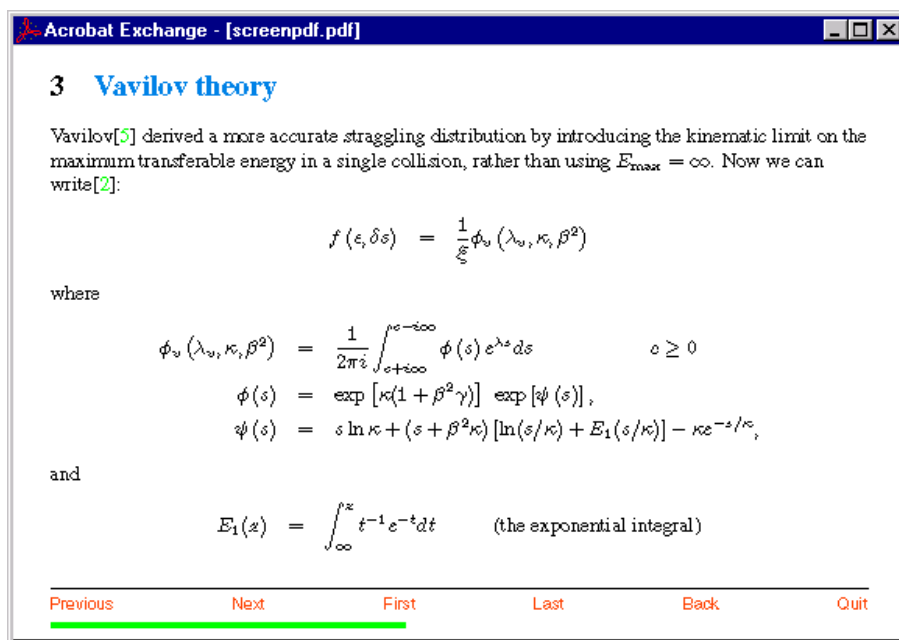


Figure 3: PDF display optimized for viewing on a screen

ically turned into hyperlinks). Web browsers can be configured to load Acrobat as a helper application and URL's in the PDF files are passed to the browser to resolve, giving a seamless integration. Thus, pages can be optimized for reading and navigating on the screen (Figure 3).

## 2.2 Down-translation to HTML

A L<sup>A</sup>T<sub>E</sub>X source document can be translated directly into HTML with tools such as LaTeX2HTML (<http://www.latex2html.org/>) or TeX4ht (<http://www.cis.ohio-state.edu/%7Eegurari/TeX4ht/>)

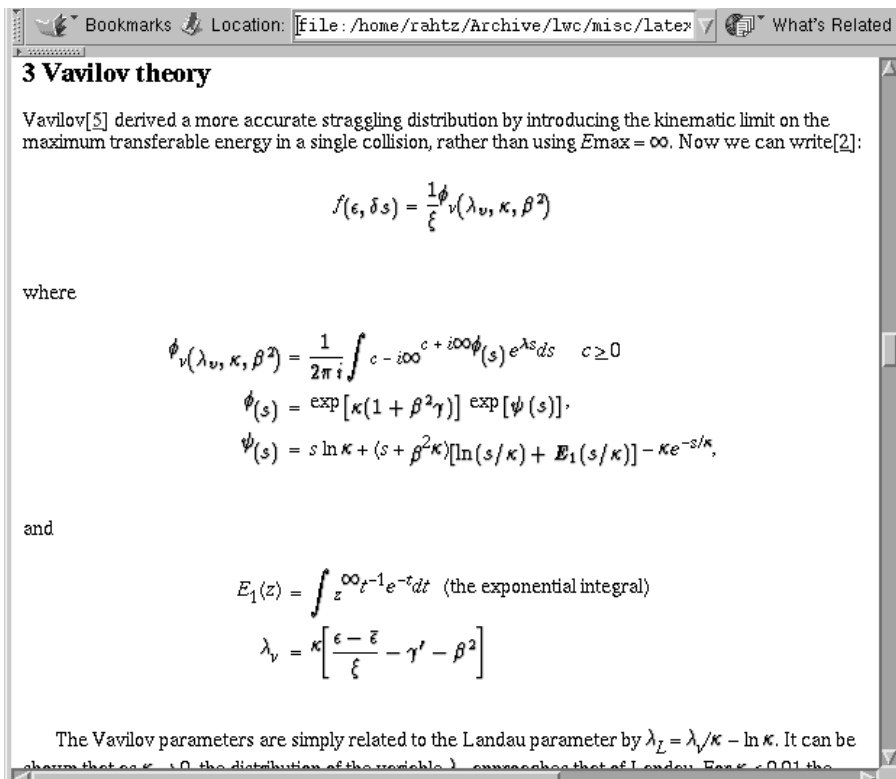


Figure 4: Conversion to HTML with math as pictures

mn.html). Even today most commonly available browsers have no built-in facilities to display mathematics properly, so that usually translation programs transform such non-standard characters (Greek, mathematical, and so on) into bitmap pictures. Figure 4 is a typical example of this approach.

With some fine tuning quite an acceptable quality can be achieved with standard browsers. Also, the cross-references, bibliographic citations, and hierarchical structure of the  $\text{\LaTeX}$  source file can be translated into HTML hypertext anchor functionality, allowing for optimal navigation and integration in the Web. Far-reaching customization is possible via command options and extension files.

On the other hand, because most non-Latin symbols are translated into bitmaps, many thousands of GIF or PNG bitmap images may be created for a scientific document of more than a few pages (depending, of course, on the complexity of the mathematical content and on whether reuse of images is allowed). Because all of these small files have to be downloaded together with the HTML source of the page, the time needed to display a page in a browser can be rather long. These images also take a lot of disk space, and often a modification in the  $\text{\LaTeX}$  source necessitates rebuilding the whole set of images.

### 2.3 Browser plug-ins

A browser plug-in, such as IBM's **techexplorer**, understands a large subset of the  $\text{\LaTeX}$  language and can thus display  $\text{\LaTeX}$  source code *directly* inside a browser. Similarly the **WebEQ** application, written in Java, can interpret another subset of the  $\text{\LaTeX}$  vocabulary. Although this approach is quite fast, a serious drawback is that these plug-ins must be downloaded and installed on all browsers that one wants to use. Moreover, a separate version is needed for each computer platform.

### 3 Integrating math in the browsers

The advent and ready availability of the personal computer drastically reduced the cost of producing, storing, and maintaining of electronic documents, also in scientific publishing where the correct markup of semantic and graphical content can be quite complex. Moreover, the World Wide Web has made sharing of electronic documents more reliable, easier, faster and a lot cheaper than in the past.

To take full advantage of the new possibilities of disseminating knowledge electronically via the Web, the Extensible Markup Language (XML) family of standards and the platform-independent Java language, have been created. They allow one to optimally exploit using a unified approach the vast amount of information stored in XML databases and present it in a user-defined fashion.

Nevertheless, even though the rise to prominence of the World Wide Web started in 1994, critics were quick to point out that the utility of the Web for scientific communication was severely limited by its lack of support for scientific, in particular, mathematical notation. Therefore, early on work started at the *World Wide Web Consortium* (<http://www.w3c.org/>), the standards body for the Web, to develop an effective framework for math on the Web, and ever since work has proceeded steadily.

Although many of the Web developments were driven by members of the scientific communities, advances in the possibilities to view math on the Web have been frustratingly slow, often implemented by ad-hoc hacks that did not work together. Therefore, from the point of view of the average author, there has been little tangible progress in support for math in mainstream browsers.

Recently, however, surfing on the wave of quickly expanding XML technologies, languages to describe mathematics (MathML), graphics (SVG), CML (chemistry), GEML (Genome expression) BIOML (biology), as well as for many others areas, such as music, finances, business, have been created.

Thanks to W3C's experimental *Amaya*, Microsoft's *Internet Explorer 6* and the upcoming *Mozilla 1.x/Netscape 7.x* browsers, the situation is improving. These browsers implement a number of these new standards-based technologies and will make better math support possible.

#### 3.1 Math on HTML platforms

The job of extending an HTML Web browser to handle math notation consists essentially of two parts: 1) encode math notation in the document, and 2) make the browser display it. The first task must be handled in a standard way, so that authors can create a single document that works on all platforms, while the second task is, of course, browser-specific.

The problem of encoding math notation in a Web document was mostly solved in 1998 with the publication of the XML and MathML Recommendations, which respectively specify a general syntax for Web documents, and a specific XML vocabulary for (presentation and content) math. Today, a coherent set of W3C publications exists that details the architecture for accommodating math and graphics in XML documents. The basis is XML, XHTML, SVG, and MathML for content, XSL (Extensible Stylesheet Language) and CSS (Cascading Style Sheets) for styling and transformation, and JavaScript and DOM (Document Object Model) for scripting of dynamic features.

#### 3.2 Varieties of mathematics on the Web

Most documents on the Web containing math fall into one of three categories: research articles, assignments and other classroom documents, or instructional expositions of a topic.

For researchers, increased accessibility of math on the Web is probably the dominant added value. Staying current and being able to find related work in a field are the critical needs of researchers. Increasingly, peer-reviewed journals are turning to the Web to provide value-added features that increase accessibility, searching and indexing, maintaining errata, forward and backward reference tracking. Consequently, from the authoring point of view, researchers need tools

that efficiently publish Web versions of print documents in ways that maximize the ease with which other researchers can find them. Furthermore, research articles are fundamentally print documents (static math), regardless of whether or not they are distributed electronically, since readers nearly universally prefer a print document when intense study and concentration is required. Therefore, authoring tools for research on the Web must make the production of very high-quality hard copy relatively easy.

A second category of math documents on the Web are related to day-to-day course work, such as assignments, quizzes, practice tests, syllabi, etc. These documents are mostly prepared simultaneously in both print and Web form, since hard copies are typically handed out in class. However, unlike research articles, the Web version exists primarily only for convenience rather than accessibility, and the system to generate the Web should be simple, light-weight, and cheap, so that it can be installed by students.

Expository math on the Web documents are much harder to quantify than research articles and course work. Compared to the other two categories, there are fewer examples, and the range of approaches is extremely varied. Nonetheless, a couple of clear authoring patterns are emerging. One is what might be called the *mathlet*, usually an applet devoted to interactively demonstrating a single concept, supported by several pages of static math exposition, often with heavy use of graphics. Another common pattern is the computer algebra notebook paradigm, where a piece of expository text has certain *live* equations within it that can be manipulated in various ways by the reader to gain a fuller understanding of the topic. Both these patterns rely heavily on dynamic math.

### 3.3 MathML tools for research

The huge majority of scientific research documents are authored in one of two ways: as Microsoft Word documents containing Equation Editor or MathType equations, or as some flavor of  $\text{T}_{\text{E}}\text{X}$ , with  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  being the most common. Within the mathematics and physics communities,  $\text{T}_{\text{E}}\text{X}$  is the dominant format, while Word is more prevalent in most other research disciplines.

In the following sections we look in more or less detail at some commercial and free tools that have some support for MathML authoring or display.

#### 3.3.1 Amaya

W3C's prototype Amaya browser can handle XHTML, the presentation part of MathML and a subset of SVG. Figure 5 shows how the XHTML, MathML, and SVG namespaces can be combined in a document (see (<http://www.w3.org/TR/XHTMLplusMathMLplusSVG>)) and have it displayed interpreting the various vocabularies it references. Amaya offers several views for easy editing and debugging of documents. Figure 6 shows the rendered window at the top left, while in the middle part one has a hierarchical view, and below it the source that can be edited. One sees clearly the highlighted part of the text and the cursor in the display, source, and hierarchical views, which makes it easy to navigate and edit a document. Other ancillary views are the links (top right), the table of contents (second from top at the right, with the section titles), and an alternate view (third from top at the right). Finally, at the bottom left we have a window for displaying warning or error messages.

#### 3.3.2 ConTeXt and Omega

ConTeXt (<http://www.pragma-ade.com/>), a free program developed by Hans Hagen, is based on  $\text{T}_{\text{E}}\text{X}$  and can convert MathML to PDF or DVI files. An on-line testing page (<http://216.109.141.43/mathml.htm>) and an example (<http://cod.pragma-pod.com/>) for printing on demand are available. It has also a special mode for typesetting XML (MathML) directly.

Omega (<http://omega.cse.unsw.edu.au:8080/index.html>), is a modified and extended version of the  $\text{T}_{\text{E}}\text{X}$  engine itself, which has been extended to 31 bits for characters, tokens, registers, fonts, etc., and provides full Unicode support at the source level. The upcoming release Omega 2

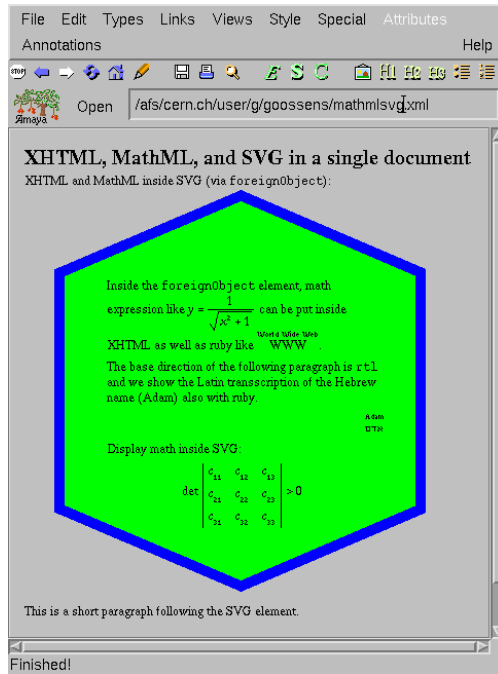


Figure 5: Amaya displaying document containing XHTML, MathML, SVG

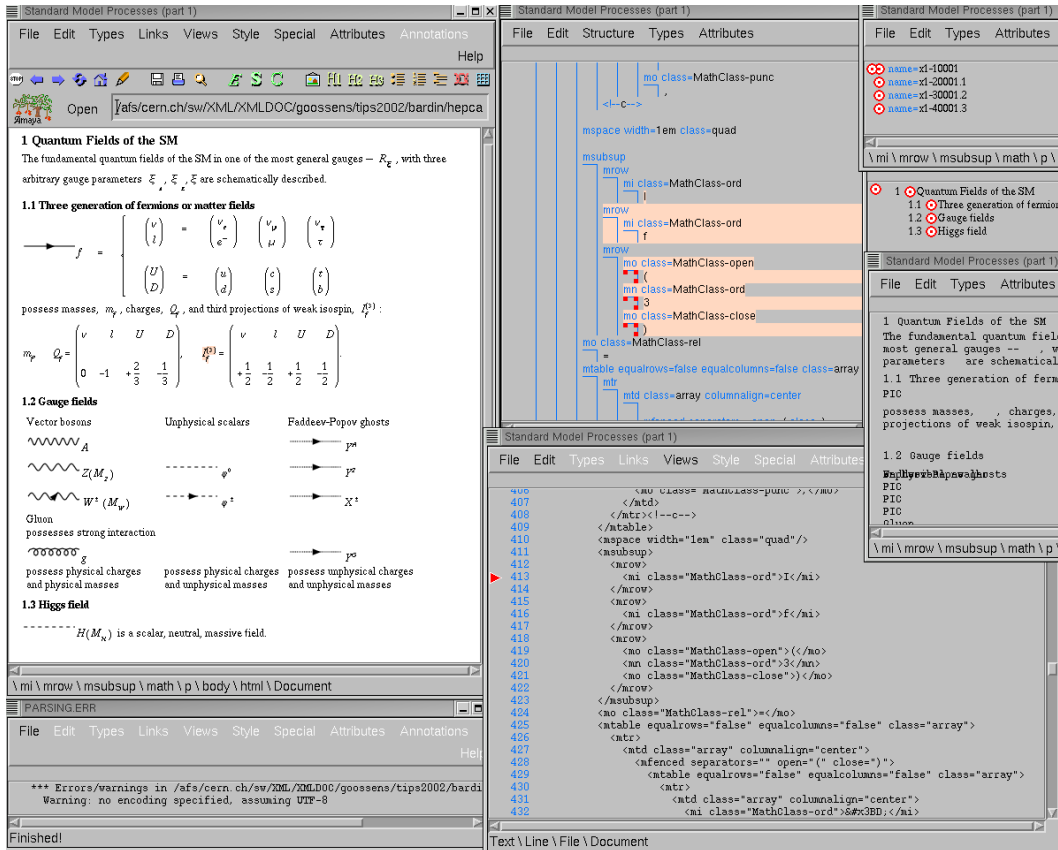


Figure 6: Editing with Amaya

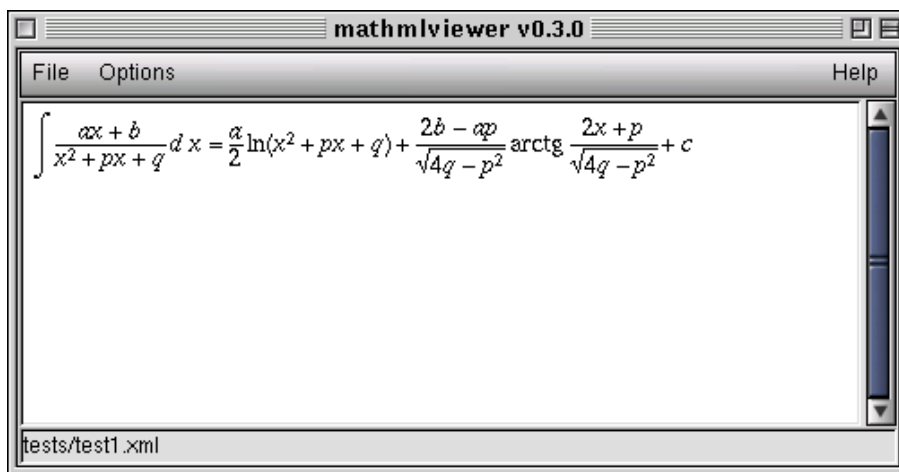


Figure 7: Viewing math with the GtkMathView widget

concentrates on micro-typographic issues and defines standard interfaces for switching typesetting environments, fonts, etc. The next release **Omega 3** will deal with the complexities of macro-typography, including multiple input sources (critical editions, etc) and multiple output formats (XML, MathML, PDF, ...). The present version already has some experimental provisions to deal with MathML, both at input or at output level.

### 3.3.3 GtkMathView

**GtkMathView** (<http://www.cs.unibo.it/helm/mml-widget/>) is an open source widget that can render Presentation MathML markup. It can be used in any software based on the GTK+ (<http://www.gtk.org>) toolkit. **GtkMathView** is a standalone, light-weight component and not a full browser. GTK applications can use the widget as a window for displaying mathematical formulas and doing simple interactions. Among other features, **GtkMathView** includes support for breaking long mathematical expressions, rendering of stretchy operators, and provides a customizable support for additional fonts. The tool is made of three main components: a portable rendering engine for MathML written in C++, the GTK interface, and a PostScript interface which renders MathML documents to encapsulated PostScript using **TeX** fonts. Figure 7 shows **GtkMathView** at work.

### 3.3.4 IBM techexplorer

**IBM techexplorer** (<http://www.ibm.com/software/network/techexplorer>) is a viewer for technical and scientific documents. **IBM techexplorer** comes as a plug-in for **Netscape Navigator** and **Microsoft Internet Explorer**, with for the latter support for XML Behavior, and an **ActiveX** control for applications like **Microsoft PowerPoint** and **Word**. **techexplorer** can handle documents marked up in **TeX**, **L<sup>A</sup>TeX**, and **MathML**. It includes full support for **MathML 2.0**, augmented **L<sup>A</sup>TeX** display, ways to enliven documents via scripting/programming and a **Web equation editor**, and connectivity with **Mathematica**. A version for **Macintosh** exists. A free *introductory* and a commercial enhanced *professional* version are available.

### 3.3.5 MacKichan Software

**MacKichan Software** (<http://www.mackichan.com/index.html>)'s **Scientific Workplace** supports XML and **MathML** from version 4.0 onward by allowing one to export a document to **HTML** format with the mathematics rendered as graphics or as **MathML**.

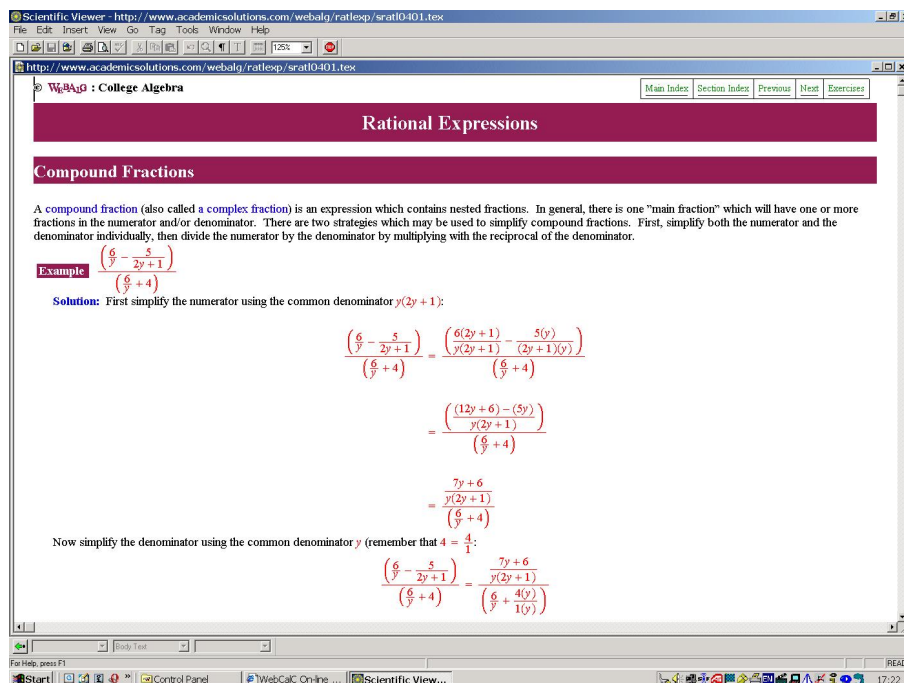


Figure 8: The WebMath Project

Documents produced with one of MacKichan Software's mathematical word processors can be read with the freely available **Scientific Viewer** (<ftp://ftp.mackichan.com/download/version40trials/sviewer400.exe>) application (Windows only). Thus, authors can create a complete Web that displays mathematics correctly and cleanly. The documents on the Web can include graphics and hyperlinks; together, they can create a rich and beautiful Web.

An example of such a Web project produced with this software is the **WebMath Project** (<http://www.math.tamu.edu/~webcalc>) at Texas A&M University. Material from the algebra course as displayed by the **Scientific Viewer** on Windows is displayed in Figure 8.

Advantages of the **Scientific Viewer** approach is that there are no limits to the mathematics that can be displayed. Files are small, download quickly, and are created with  $\text{T}_{\text{E}}\text{X}$  which is well known in the scientific community. Since the mathematics has not been converted to bitmap graphics, it will not appear dotted or grainy when it is printed. The solution is relatively inexpensive for the author and free for the readers. Moreover, since links to HTML files and links to  $\text{T}_{\text{E}}\text{X}$  files can be intermixed users can access both kinds of files. On the other hand, readers must run Microsoft Windows and download and install the viewer (a large file). Moreover, it is not an XML solution, since MathML is not used in this specific browser.

From versions 4 onward the XHTML exported by **Scientific Workplace**, **Scientific Word**, and **Scientific Notebook** can contain mathematics rendered as MathML or in various graphics formats, such as .bmp, .gif, .jpg, or .png, for non-MathML aware browsers. In this case the generated output files can be viewed with all browsing tools on most computer platforms and any XML editing tool can be used.

### 3.3.6 Maple

**Maple** (<http://www.maplesoft.com/main.html>) is a symbolic and numeric computation system with support for importing and exporting MathML2.0, including both presentation and content forms of MathML (not yet tested since I have no Maple at CERN).

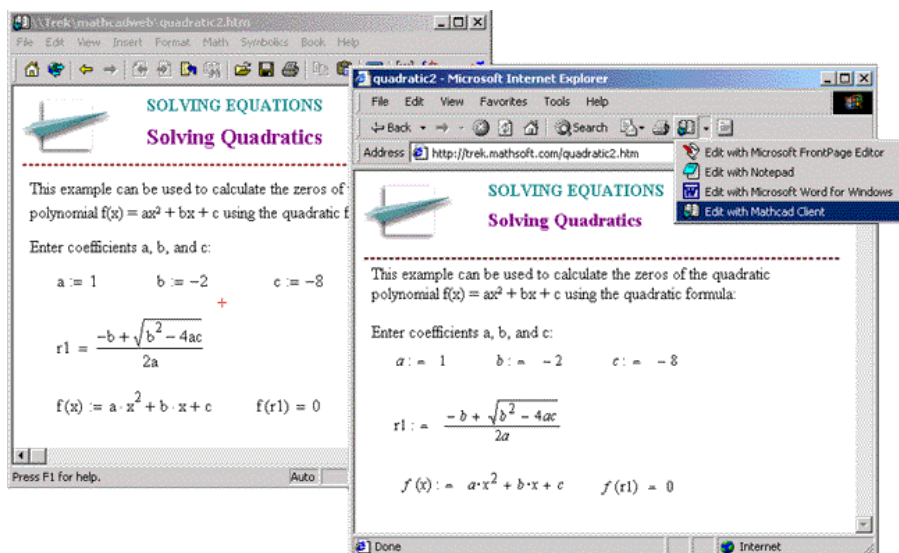


Figure 9: The Mathcad tool

### 3.3.7 Mathcad

Mathcad (<http://www.mathcad.com>), by Mathsoft Engineering & Education, Inc., is a leading calculation and technical documentation tool for professional and educational users. Mathcad uses standard math notation as its interface. Files can be saved as dual-purpose Web documents, incorporating both presentation and semantic forms of MathML. They can be viewed on the Web using IBM *techeplorer* and re-imported from the Web by Mathcad as live mathematical documents. In Figure 9 the same document is shown in Mathcad and a Web browser equipped with IBM *techeplorer*. Mathcad was launched directly from the browser for further editing and calculation of the MathML document.

### 3.3.8 Mathematica

Mathematica (<http://www.wolfram.com>) is a technical computing system with high-quality mathematical typesetting and editing. It is a visual typesetting and authoring tool which both renders and exports MathML. Figure 10 displays various formats in which the content of a Mathematica notebook can be saved, in particular HTML and MathML.

### 3.3.9 MathML to SVG

SchemaSoft has created software called Custard (<http://www.schemasoft.com/MathML/>) that converts from MathML to SVG. It runs on Windows, Mac OS X, and Linux or any machine that has Java support. The program includes a Java executable to convert from MathML 2.0 to SVG 1.0 using XSLT style sheets, documentation, a set of examples, and an HTML-based viewer for displaying the examples.

Adrian Frischauf developed the open source *dvi2svg* (<http://www.ags.uni-sb.de/~adrianf/dvi2svg/>) Java program that transforms a DVI file without PostScript inclusions into SVG. It uses the SVG version of the Computer Modern fonts described later.

### 3.3.10 MS Word and MathType

As already mentioned a large fraction of scientific documents are authored in Word. For documents that require math notation, the only real choices are to use the Equation Editor included with

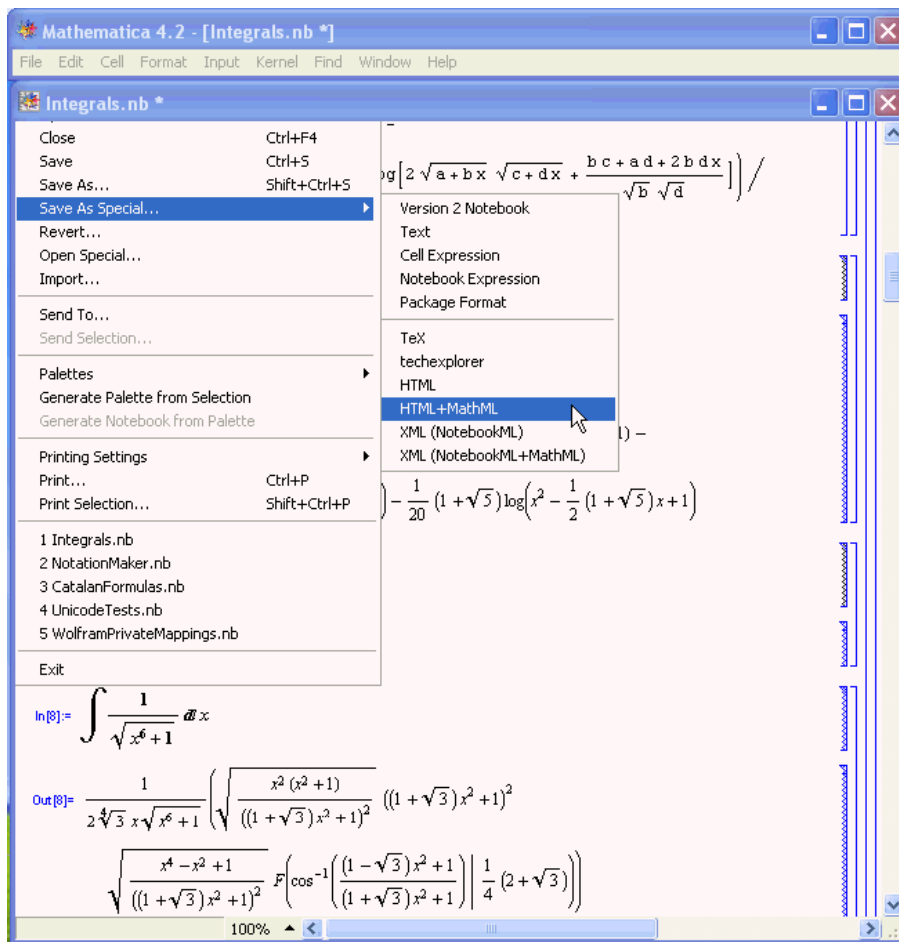


Figure 10: The Mathematica system at work

Word or to upgrade to **MathType**, the professional version of **Equation Editor**, that is an almost essential tool for **Word** authors who make heavy use of mathematical notation or need Web output.

**Mathtype** (<http://www.mathtype.com>) by Design Science is an interactive tool that generates mathematical notation for all types of print and Web-based technical documents. **Mathtype**'s built-in MathML translator converts mathematical expressions created with **Mathtype** and **Equation Editor**, a junior version of **Mathtype** that comes with **Microsoft Office**, **Corel WordPerfect** and many other Windows and Macintosh products), to MathML.

To address the many problems of **Word** output **Mathtype** (<http://www.mathtype.com>) by Design Science adds a powerful interactive tool that generates mathematical notation for all types of print and Web-based technical documents by selecting from a number of export options in a panel. An important export configuration option is the choice between generating HTML plus MathML, or generating HTML plus GIF images.

**MathType** generates presentation MathML using a rule-driven translator mechanism. The rule sets are ordinary text files that sophisticated authors can customize to tweak the MathML being generated. With **MathType 5** authors can choose between nine MathML target platforms, each of which generates the extra glue code or document declarations required by specific add-on components, such as **MathPlayer**, **WebEQ Viewer Control**, or **techemplorer**, or the MathML-enabled browsers **Mozilla/Netscape** and **Amaya**. Once you select a translator, expressions with **MathType** expressions can be copied and pasted, dragged and drop into a MathML document; **MathType** will convert the equations to MathML on the fly, as shown in the example in Figure 11.

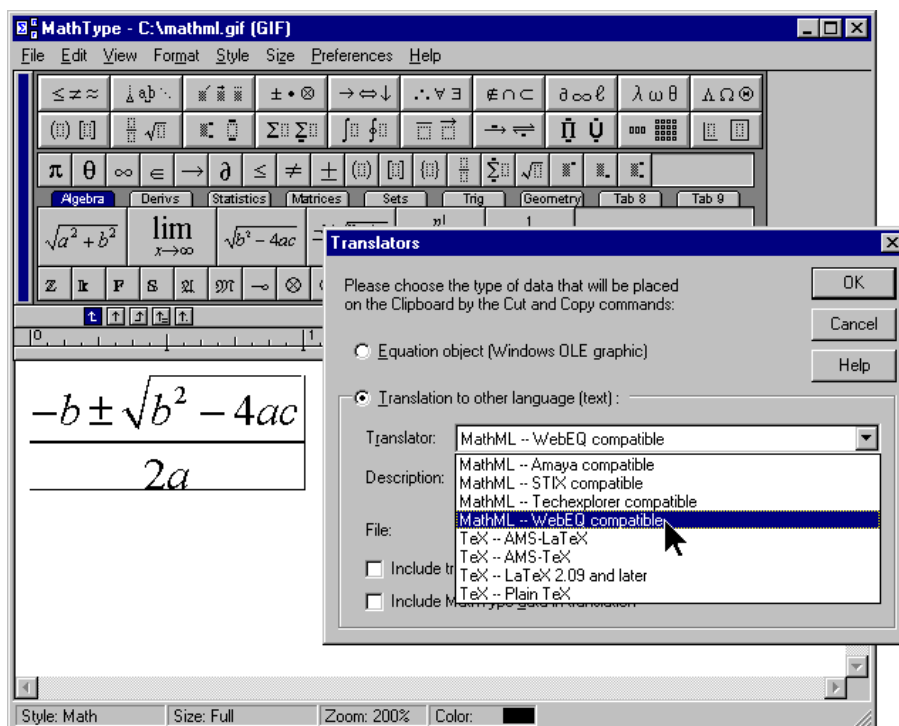


Figure 11: Export of an equation in a given format with Mathtype

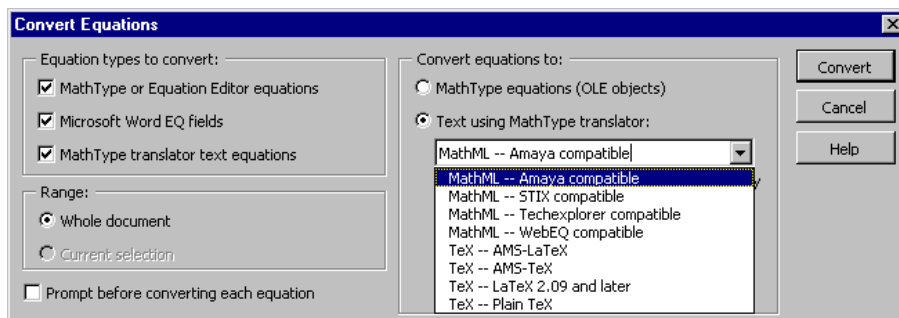


Figure 12: Converting all equations in a Microsoft Word into MathML

MathType has the capability of converting all the MathType, Equation Editor and Microsoft WordEQ fields in a Word document to MathML, with a single command (Figure 12).

Microsoft recently introduced a new technology called *Behaviors* (<http://msdn.microsoft.com/library/default.asp>), also known as *dynamic HTML*, which allows low-level integration between an add-on component and Internet Explorer 6 on Windows. With *Behaviors*, it is possible to write add-on browser components that eliminate the earlier problems with alignment, sizing and printing. Using the new Behavior technology, Design Science (<http://www.dessci.com/>) has developed a new MathML rendering component called *MathPlayer*, which can be downloaded freely. Thus, for Internet Explorer users on Windows *MathPlayer* provides a fast and seamless MathML rendering tool

Figure 13 is a composed image with at the top right a test page indicating that *MathPlayer* version 1.0 beta 3 was successfully installed on Internet Explorer. At the left is a example page showing how *MathPlayer* displays MathML markup, with at the lower right of the image the XHTML/MathML source corresponding to the first displayed equation ([1]). The page was

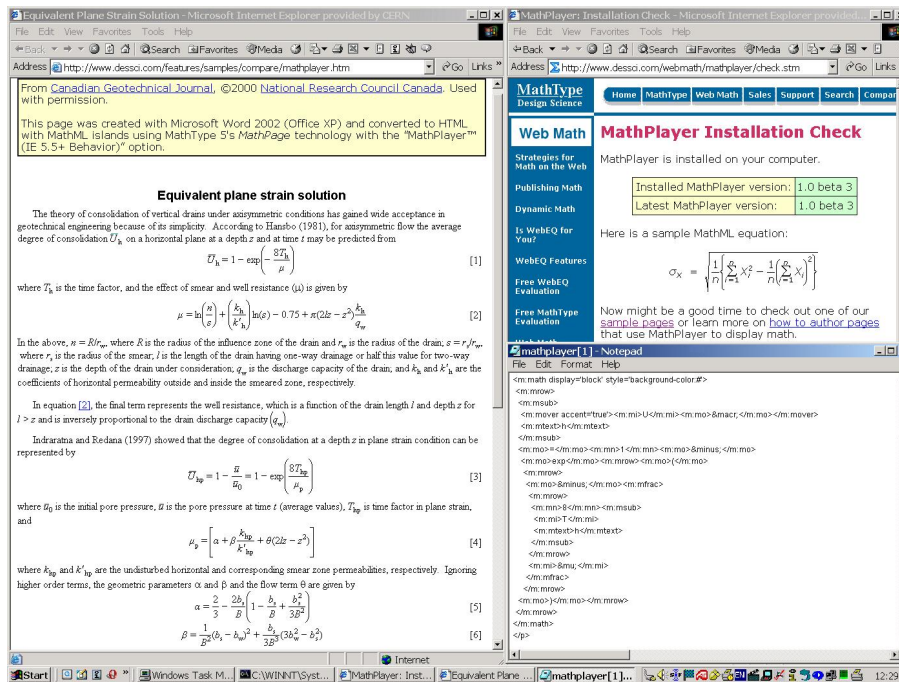


Figure 13: MathPlayer at work

generated with Microsoft Word 2002 and was converted to HTML with MathML islands using one of the translation options of MathType (as explained previously).

### 3.3.11 Mozilla

Mozilla (<http://www.mozilla.org/>) is an open-source Web browser, designed for standards compliance, performance and portability. Coordination of development and testing of the browser is provided by discussion forums, software engineering tools, releases and bug tracking.

Recent versions of the Mozilla browser, in particular version 1 and beyond, include native support for displaying Presentation MathML in Web pages (see the MathML project page (<http://www.mozilla.org/projects/mathml/>)). This open source browser is available for all major platforms and can also be compiled to include (partial) SVG support, and several other Web technologies. Figure 14 shows the same page containing MathML markup as shown in Figure 6. This proves the portability of well-written XHTML and MathML between various MathML-enabled browsers. Netscape 7.x has the same MathML support as Mozilla.

### 3.3.12 ORCAA MathML research

The Ontario Research Center for Computer Algebra Group (ORCAA) is working on various research projects related to MathML (<http://www.orcca.on.ca/MathML/>) and, more generally, XML technologies for the communication of mathematical content over the Internet and among applications.

ORCAA already offers an online  $\text{T}_{\text{E}}\text{X}$  to MathML Translator (<http://www.orcca.on.ca/MathML/texmml/textomml.html>), where customized relationships between MathML and  $\text{T}_{\text{E}}\text{X}$  elements can be specified in a map file. Another project is the design and implementation of an architecture for typesetting mathematics encoded in MathML, in particular the problem of rendering the same markup on a variety of different target media, each with different, peculiar rendering capabilities, and distinct levels of editing, quality, interactivity, etc. A MathML to  $\text{T}_{\text{E}}\text{X}$  translator is available online (<http://www.orcca.on.ca/MathML/texmml/mmltotex.html>).

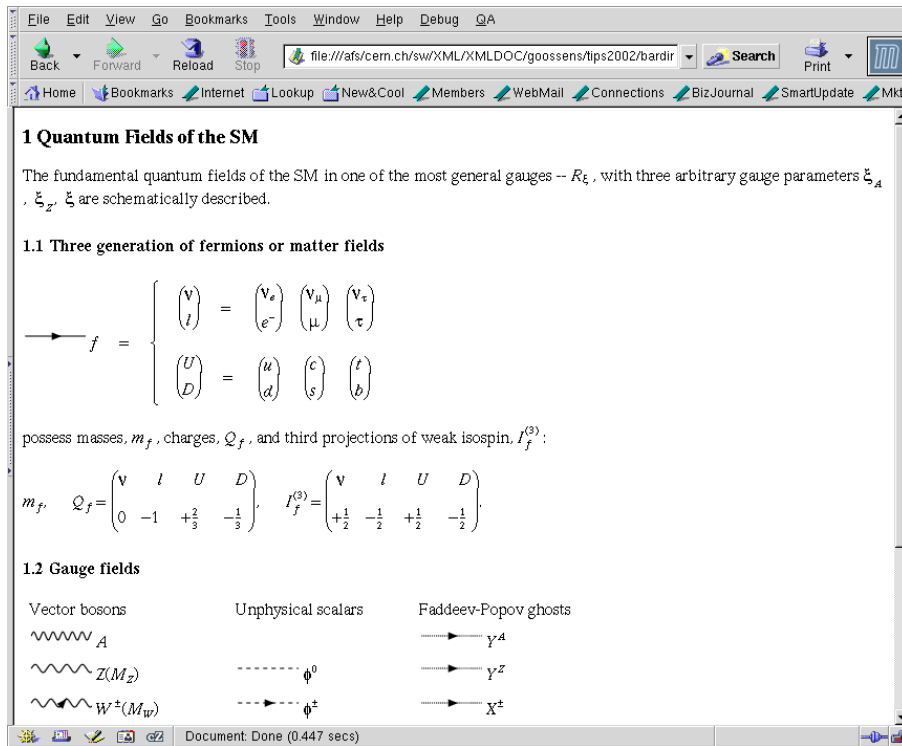


Figure 14: MathML viewed with Mozilla

Work is also going on in the area of the definition of a Schema for MathML, and a study of the way how to communicate mathematics knowledge between applications, in particular the use of presentation and content MathML markup as external resource format.

### 3.3.13 Publicon

Wolfram Publicon (<http://www.publicon.com>) is a full-featured technical authoring system from Wolfram Research designed to simplify the creation of publication quality structured documents and is to be released mid-2002. It incorporates the same unique visual authoring tool for mathematical typesetting as Mathematica, along with a point-and-click interface for managing notes, citations, and other document features of typical academic and scientific papers. Publicon documents convert automatically to XML with MathML (NotebookML and other XML types), HTML (with or without MathML), HTML with CSS (for MS Word), L<sup>A</sup>T<sub>E</sub>X (including support for the AMS and PhysRev packages), and IBM techexplorer. A free Windows/Mac version limited to the math typesetting system is currently available.

### 3.3.14 TeX4ht

TeX4ht (<http://www.cis.ohio-state.edu/%7Eegurari/TeX4ht/mn.html>) is probably the most powerful and sophisticated T<sub>E</sub>X to HTML (plus MathML) converter currently available. It uses a kind of hybrid approach, first performing analysis at the T<sub>E</sub>X source level which it uses to insert hints into the DVI file using special commands. It then processes the DVI output to generate the final document. TeX4ht is highly configurable and can be used to generate output in a wide variety of XML dialects in addition to HTML. The advantage is superior output, but the disadvantage is a fairly steep learning curve. It has been extensively used in the research related to the TIPS project to study the translation of L<sup>A</sup>T<sub>E</sub>X to XML vocabularies (XHTML and presentation MathML). TeX4ht is open source.

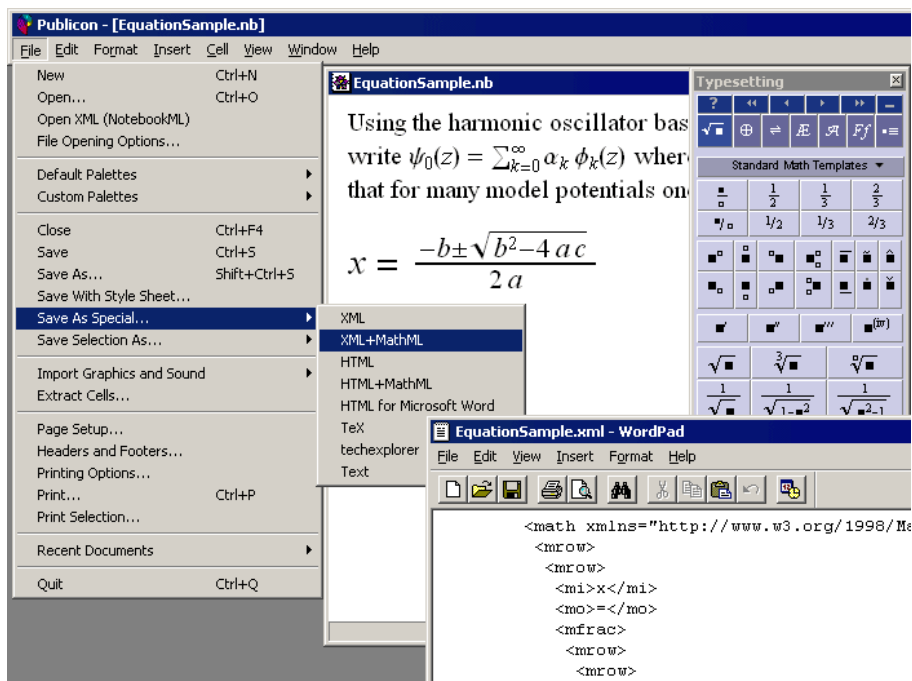


Figure 15: MathML viewed with Wolfram Publicon

### 3.3.15 TtM, a TeX to MathML translator

TtMML (<http://hutchinson.belmont.ma.us/tth/mml>) is Ian Hutchinson's powerful (La)TeX to HTML+MathML translator, which is still under active development (Windows and free Linux version). The XHTML/MathML version of the physics paper shown in Figure 18 was translated from L<sup>A</sup>T<sub>E</sub>X using TtMML. The quality and completeness is not (yet) as good as with TeX4ht, but the processor runs extremely fast.

### 3.3.16 WebEQ Developers Suite

With HTML the usual way of handling interactivity not involving computation is to dynamically modify the document by using script code embedded in the page, triggered by the user clicking on buttons, entering text, etc. MathML handles interactivity with *MathML actions*, which are encoded directly in the equation markup. Although there is already fairly extensive support in browsers and add-on rendering software for interactivity, authoring dynamic math using these techniques requires skill with programming and a strong background in Web development. The recently released **WebEQ 3 Developers Suite** makes authoring dynamic math somewhat easier and is an interesting tool to explore.

The **WebEQ Developers Suite** (<http://www.dessci.com/webmath/webeq/features.stm>) is actually a collection of five tools, **WebEQ Editor**, for authoring presentation and content MathML, **WebEQ Publisher**, for processing HTML pages containing math markup, **WebEQ Input Control**, which functions as an easy-to-use graphical equation editor in a Web page, **WebEQ Viewer Control**, that displays MathML in any Java-capable browser, **WebEQ Equation Server**, which works behind the scenes to facilitate batch processing and processing via scripts on a server.

**WebEQ Editor** gives authors a graphical way to insert MathML actions into equations. MathML actions trigger one of a handful of dynamic behaviors when a reader moves the mouse over a part of an equation or clicks on it (see Figure 16).

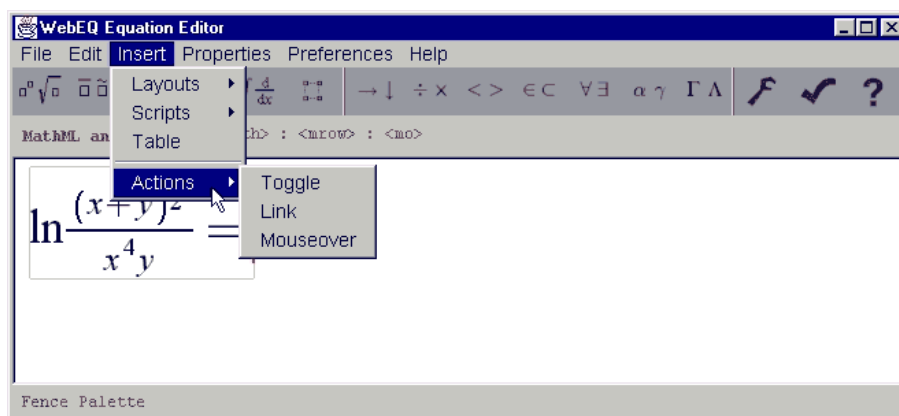


Figure 16: MathML viewed with WebEQ

### 3.3.17 webMathematica

webMathematica (<http://www.wolfram.com/products/webMathematica/>) is an application to create Web sites that allow users to compute and visualize results directly from a Web browser. Based on Java Servlet technology, webMathematica is fully compatible with Mathematica and other dynamic Web systems. webMathematica allows the generation of dynamic content by incorporating technologies such as HTML, MathML, and Java applets. With nothing more than knowledge of HTML and Mathematica, you can create custom interfaces to mathematical based Web sites. MathML functionality, which is built into Mathematica itself, can be accessed through webMathematica. Examples of converting Mathematica code into MathML and rendering MathML code via webMathematica are available online (<http://www.wolfram.com/products/webmathematica/examples/>).

## 3.4 The Universal Math Stylesheet

Given the advances in rendering software and coding standards, only one obstacle to ubiquitous and effective math support remains: different rendering technologies require bits of *glue code* to signal the browser how to handle the MathML equations it might encounter in a document. In some cases, this extra code takes the form of special declarations in the document header. In others, special wrapper code is required around each equation. In still other cases, a little code is required in both places. On the surface, this would seem to make it impossible for an author to publish a single document that simultaneously works in all rendering environments.

The solution envisioned on HTML platforms is a standardized way of transforming parts of a document on the fly according to rules in an XSLT (<http://www.w3.org/TR/xslt>) (XSL Transformations) stylesheet. XSLT rules can take into account what browser is being used to view the page, and what add-on rendering components are installed. This enables authors to ignore the *glue code* that used to be necessary to fire up a specific rendering component to handle math notation. Instead, authors generate documents which are strictly standards-compliant, and at run time, the stylesheet running in the reader's browser adds whatever glue code is necessary to render MathML based on what is installed on the reader's system.

Internet Explorer 6 and Netscape 6 are the first browsers to fully implement XSLT. Therefore, as explained above, the W3C Math Working Group developed an XSLT-based *Universal Math Stylesheet* (<http://www.w3.org/Math/XSL/>), mainly written by David Carlisle (NAG, member of the L<sup>A</sup>T<sub>E</sub>X development team and one of the editors of the MathML 2.0 Specification).

The *Universal Math Stylesheet* searches through a list of possible rendering configurations and uses the first one that matches the reader's system. Authors can customize the order of the search, to specify a preferred rendering configuration on systems that have more than one available. In general, the stylesheet attempts to use native implementations and add-on renderers first. If that

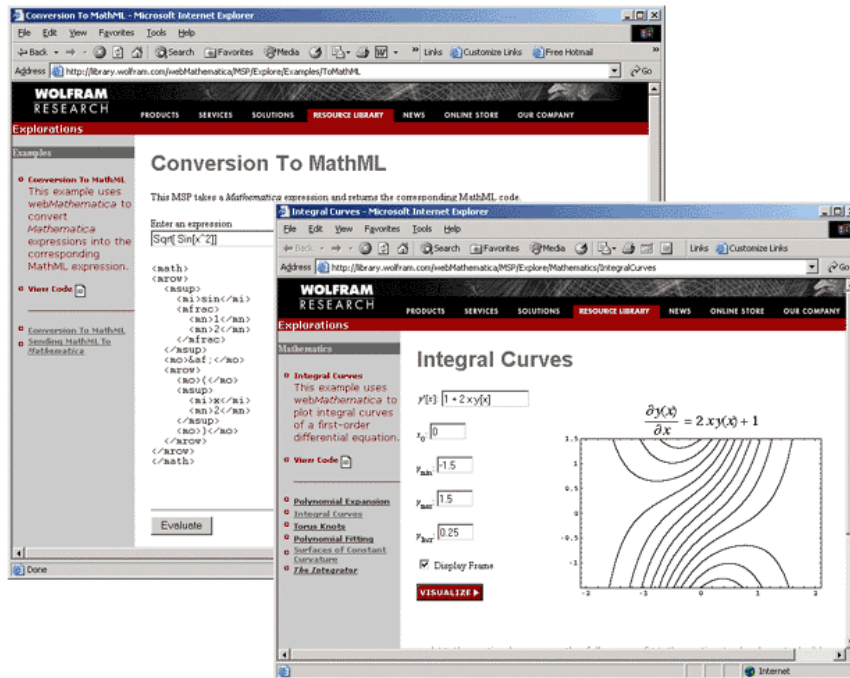


Figure 17: MathML viewed with webMathematica

fails, it will generate HTML/CSS/JavaScript code on the fly to approximate traditional math layout in 6.x browsers.

The math rendered by the stylesheet ranges from crude but legible to very high quality depending on the combination of browser, operating system and add-on software. In any case, and for the first time, the *Universal Math stylesheet* guarantees that authors can be relatively certain that most readers will actually be able to see their MathML equations rendered intelligibly in a Web page.

Figure 18 shows how four browsers, **Internet Explorer 6** at the top left, **Opera 6.01** at the bottom left, **Mozilla 1.0** at the top right, and **Amaya 6.1** at the bottom right, interface with the *Universal Math stylesheet*. An identical part of a physics document is displayed in each case. **Mozilla 1.0** has native MathML support and shows the contents nicely, while **Amaya** does also a fairly good job, although it has some characters missing. On the other hand, **Internet Explorer** (without any math plug-in) falls back on the CSS treatment of the style-sheet, while **Opera** shows only the characters on a single line. It is clear that it would be difficult to read a math paper with the latter application, but in all other cases the presentation is good to acceptable.

## 4 Mathematics and Unicode

XML's basic character encoding is Unicode, so that one can use directly the many math characters that are present in the various Unicode planes. In particular, with the publication of Unicode 3.2 (<http://www.unicode.org/unicode/reports/tr28/>) in March 2002, quite a complete set of standard math alphabets has been added so that today a total of 1927 mathematical symbols are available. This repertoire is the result of input from many sources, notably from the STIX Project (Scientific and Technical Information Exchange), a cooperation of mathematical publishers. The STIX collection includes, but is not limited to, symbols gleaned from mathematical publications by experts from the American Mathematical Society (AMS) and symbol sets provided by Elsevier Publishing and by the American Physical Society. The new repertoire enables the display of

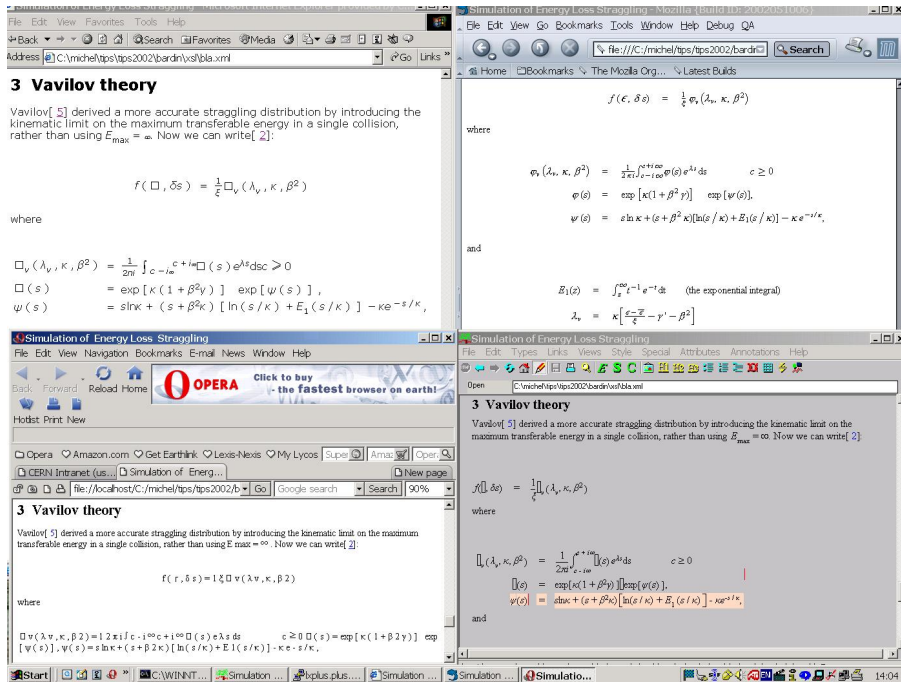


Figure 18: Four browsers with the Universal Math stylesheet

virtually all standard mathematical symbols. Nevertheless this work must remain incomplete; mathematicians and other scientists are continually inventing new mathematical symbols and the plan is to add them as they become accepted in the scientific communities. A detailed description of how Unicode can be optimally combined with math is described in the document Unicode Support for Mathematics (<http://www.unicode.org/unicode/reports/tr25/>).

## 4.1 Mathematical characters and alphabets

The mathematical characters in Unicode are distributed over various blocks, as shown in Table 1, where blocks flagged with \* contain non-mathematical characters as well. Moreover, some other characters for occasional use are present in non-listed blocks.

Mathematics often needs a number of Latin and Greek alphabets that on first sight seem merely to be font variations of one another. For example the letter H can appear as plain or upright (**H**), bold (**H**), italic (*H*), and script (*H*). However in any given document, these characters have distinct, and usually unrelated *mathematical semantics*. Thus, a normal H represents a different variable from a bold **H**, etc. If these attributes are dropped in plain text, the distinctions are lost and the meaning of the text is altered. By encoding a separate set of alphabets, it is possible to preserve such distinctions in plain text. Moreover, sometimes two variants of the same Greek character are included. This is because they can appear in the same mathematical document with different meanings, even though they would have the same meaning in Greek text.

The *Mathematical Alphanumeric Symbols* block (last line in Table 1) were added to plane 1 in Unicode 3.2. It contains a large extension of letterlike symbols used in mathematical notation, typically for variables. The characters in this block are intended for use only in mathematical or technical notation; they are not intended for use in non-technical text. When used with markup languages, for example with MathML, the characters are expected to be used directly, instead of indirectly via entity references or by composing them from base letters and style markup. An overview of these alphanumeric symbols encountered in mathematics is given in Table 2, where alphabets on lines marked with \* have characters also in the BMP (*Basic Multilingual Plane* or plane 0, range U+0000 to U+FFFF).

Table 1: Locations of Mathematical Characters

Block Name	Range	Characters
Basic Latin	U+0021 to U+007E	Variables, operators, digits*
Greek	U+0370 to U+03FF	Variables*
General Punctuation	U+2000 to U+206F	Invisible operators*
Letterlike Symbols	U+2100 to U+214F	Variables*
Arrows	U+2190 to U+21FF	Arrows, arrow-like operators
Mathematical Operators	U+2200 to U+22FF	Operators
Miscellaneous Technical Symbols	U+2300 to U+23FF	Braces, operators*
Geometrical Shapes	U+25A0 to U+25FF	Symbols
Misc. Mathematical Symbols-A	U+27C0 to U+27EF	Symbols and operators
Supplemental Arrows-A	U+27F0 to U+27FF	Arrows, arrow-like operators
Supplemental Arrows-B	U+2900 to U+297F	Arrows, arrow-like operators
Misc. Mathematical Symbols-B	U+2980 to U+29FF	Braces, symbols
Suppl. Mathematical Operators	U+2A00 to U+2AFF	Operators
Mathematical Alphanumeric Symbols	U+1D400 to U+1D7FF	Variables and digits

Table 2: Mathematical Alphabets

Math Style	Characters from Basic Set	Location
plain (upright, serifed)	Latin, Greek and digits	BMP (Plane 0)
bold	Latin, Greek and digits	Plane 1
italic	Latin and Greek	Plane 1*
bold italic	Latin and Greek	Plane 1
script (calligraphic)	Latin	Plane 1*
bold script (calligraphic)	Latin	Plane 1
Fraktur	Latin	Plane 1*
bold Fraktur	Latin	Plane 1
double-struck	Latin and digits	Plane 1*
sans-serif	Latin and digits	Plane 1
sans-serif bold	Latin, Greek and digits	Plane 1
sans-serif italic	Latin	Plane 1
sans-serif bold italic	Latin and Greek	Plane 1
monospace	Latin and digits	Plane 1

## 4.2 More transparent scientific notation with Unicode

### 4.2.1 Simplifying math notations

When typing math in  $\text{\TeX}$  (or another math manipulation program) entering, or even reading, long equations can become quite tedious. As an example consider the following typeset equation, and its  $\text{\TeX}$  source.

$$W_{\delta_1 \rho_1 \sigma_2}^{3\beta} = U_{\delta_1 \rho_1}^{3\beta} + \frac{1}{8\pi^2} \int_{\alpha_1}^{\alpha_2} d\alpha_2' \left[ \frac{U_{\delta_1 \rho_1}^{2\beta} - \alpha_2' U_{\rho_1 \sigma_2}^{1\beta}}{U_{\rho_1 \sigma_2}^{0\beta}} \right]$$

```
\[W^{3\beta}_{\delta_1\rho_1\sigma_2}
= U^{3\beta}_{\delta_1\rho_1} + \frac{1}{8\pi^2}
\int_{\alpha_1}^{\alpha_2} d\alpha_2' \left[
\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}
{U^{0\beta}_{\rho_1\sigma_2}} \right] \]
```

This should be compared to its “Unicode T<sub>E</sub>X variant” or the plain text version shown below. Typing math for T<sub>E</sub>X (or for use in a computer algebra system) becomes more readable if one directly uses Unicode characters and sees their graphical representations. Explicit precedence rules could further eliminate the need for some brace pairs in the text version.

$$\int_{-\{\alpha_1\}^{\{\alpha_2\}}} d\alpha_2' \left[ \frac{U^{\{3\beta\}\{\delta_1\rho_1\}} + \{1/8\pi^2\} U^{\{2\beta\}\{\delta_1\rho_1\}} - \alpha_2' U^{\{1\beta\}\{\rho_1\sigma_2\}}}{U^{\{0\beta\}\{\rho_1\sigma_2\}}} \right]$$

$$W^{\{3\beta\}\{\delta_1\rho_1\sigma_2\}} = U^{\{3\beta\}\{\delta_1\rho_1\}} + \{1/8\pi^2\} \int^{\{\alpha_1\}}_{\{\alpha_2\}} d\alpha_2' \left[ \frac{U^{\{2\beta\}\{\delta_1\rho_1\}} - \alpha_2' U^{\{1\beta\}\{\rho_1\sigma_2\}}}{U^{\{0\beta\}\{\rho_1\sigma_2\}}} \right]$$

#### 4.2.2 Use of Unicode math characters in computer programs

It can be quite useful to be able to type typical mathematical symbols in computer programs. A key point is that the compiler should display the desired characters in both edit and debug windows. A preprocessor can translate MathML, for example, into C++, but it will not be able to make the debug windows use the math-oriented characters unless it can handle the underlying Unicode characters. Java has made an important step in this direction by allowing Unicode variable names. The mathematical alphanumeric symbols allow this approach to go further with relatively little effort for compilers.

The advantages of using the Unicode plain text in computer programs are at least threefold.

1. Many formulas in document files can be programmed simply by copying them into a program file and inserting appropriate multiplication dots, thus reducing coding time and errors.
2. The use of the same notation in programs and the associated journal articles and books leads to an unprecedented level of self-documentation.
3. The proposed approach should also help us find out how to teach computers to understand and use arbitrary mathematical expressions.

As an example consider the following C++ program:

```
void IHBMWM(void)
{
    gammap = gamma*sqrt(1 + I2);
    upsilon = cmplx(gamma+gamma1, Delta);
    alphainc = alpha0*(1-(gamma*gamma*I2/gammap)/(gammap + upsilon));

    if (!gamma1 && fabs(Delta*T1) < 0.01)
        alphacoh = -half*alpha0*I2*pow(gamma/gammap, 3);
    else
    {
        Gamma = 1/T1 + gamma1;
        I2sF = (I2/T1)/cmplx(Gamma, Delta);
        betap2 = upsilon*(upsilon + gamma*I2sF);
        beta = sqrt(betap2);
        alphacoh = 0.5*gamma*alpha0*(I2sF*(gamma + upsilon)
            /(gammap*gammap - betap2))
            *((1+gamma/beta)*(beta - upsilon)/(beta + upsilon)
            - (1+gamma/gammap)*(gammap - upsilon)/
            (gammap + upsilon));
    }
    alpha1 = alphainc + alphacoh;
}
```

With Unicode support in the C++ compiler this can be written more readably as follows:

```
void IHBMWVM(void)
{
     $\gamma' = \gamma \cdot \sqrt{1 + I_2}$ ;
     $\upsilon = \gamma + \gamma_1 + i \cdot \Delta$ ;
     $\alpha_{mc} = \alpha_0 \cdot (1 - (\gamma \cdot \gamma \cdot I_2 / \gamma') / (\gamma' + \upsilon))$ ;
    if (! $\gamma_1$  || fabs( $\Delta \cdot T_1$ ) < 0.01)
         $\alpha_{coh} = -.5 \cdot \alpha_0 \cdot I_2 \cdot \text{pow}(\gamma / \gamma', 3)$ ;
    else
    {
         $\Gamma = 1/T_1 + \gamma_1$ ;
         $I_2 \mathcal{F} = (I_2/T_1) / (\Gamma + i \cdot \Delta)$ ;
         $\beta^2 = \upsilon \cdot (\upsilon + \gamma \cdot I_2 \mathcal{F})$ ;
         $\beta = \sqrt{\beta^2}$ ;
         $\alpha_{coh} = .5 \cdot \gamma \cdot \alpha_0 \cdot (I_2 \mathcal{F}(\gamma + \upsilon) / (\gamma' \cdot \gamma' - \beta^2))$ 
             $\times ((1 + \gamma / \beta) \cdot (\beta - \upsilon) / (\beta + \upsilon) - (1 + \gamma / \gamma') \cdot (\gamma' - \upsilon) / (\gamma' + \upsilon))$ ;
    }
     $\alpha_1 = \alpha_{mc} + \alpha_{coh}$ ;
}
```

Or even more conveniently:

```
void IHBMWVM(void)
{
     $\gamma' = \gamma \cdot \sqrt{1 + I_2}$ ;
     $\upsilon = \gamma + \gamma_1 + i \cdot \Delta$ ;
     $\alpha_{mc} = \alpha_0 \cdot \frac{1 - (\gamma \cdot \gamma \cdot I_2 / \gamma')}{\gamma' + \upsilon}$ ;
    if (! $\gamma_1$  || fabs( $\Delta \cdot T_1$ ) < 0.01)
         $\alpha_{coh} = -.5 \cdot \alpha_0 \cdot I_2 \cdot \text{pow}(\gamma / \gamma', 3)$ ;
    else
    {
         $\Gamma = 1/T_1 + \gamma_1$ ;
         $I_2 \mathcal{F} = \frac{I_2/T_1}{\Gamma + i \cdot \Delta}$ ;
         $\beta^2 = \upsilon \cdot (\upsilon + \gamma \cdot I_2 \mathcal{F})$ ;
         $\beta = \sqrt{\beta^2}$ ;
         $\alpha_{coh} = .5 \cdot \gamma \cdot \alpha_0 \cdot \frac{I_2 \mathcal{F}(\gamma + \upsilon)}{\gamma' \cdot \gamma' - \beta^2} \times \left( \left( 1 + \frac{\gamma}{\beta} \right) \cdot \frac{\beta - \upsilon}{\beta + \upsilon} - \left( 1 + \frac{\gamma}{\gamma'} \right) \cdot \frac{\gamma' - \upsilon}{\gamma' + \upsilon} \right)$ ;
    }
     $\alpha_1 = \alpha_{mc} + \alpha_{coh}$ ;
}
```

### 4.3 The advantages of Unicode in scientific texts

It is clear that thanks to Unicode mathematical expressions can usually be represented with a readable Unicode plain-text format. The text consists of combinations of operators and operands. To reveal the operators, operator-aware editors could be instructed to display operators with a different color or some other attribute. To simplify the notation, operators are assigned precedence values that control the association of operands with operators unless overruled by parentheses.

Heuristics can be applied to the Unicode plain text to recognize what parts of a document are mathematical expressions. This allows the Unicode plain text to be used in a variety of ways, including in technical document preparation, symbolic manipulation, and numerical computation.

The heuristics given for recognizing mathematical expressions work well, but they are not infallible. An effective use of the heuristics would be as an autoformatting wizard that delimits what it thinks is mathematics with mathematics on/off codes. The user can overrule incorrect choices. Once marked unequivocally as mathematics, export to MathML, compilers, and other consumers of mathematical expressions is straightforward. Thus, a workable plain-text encoding of mathematics that looks very much like mathematics exists, even when display capabilities are limited. Appropriate display software can make it look like the *real math*.

## 5 Scalable Vector Graphics

### 5.1 Introduction

As the Web has grown in popularity and complexity, users and content providers wanted ever better and more precise graphical rendering, as well as dynamic Web sites. Drop shadows, rudimentary animations, and low-resolution GIF or PNG images are ubiquitous on today's Web pages, but that technology is not really scalable. Therefore, in September 2001, thanks to a collaborative effort between many of the biggest players in the computer world over a period of more than two years in the framework of the W3C Graphics activity the *SVG (Scalable Vector Graphics)* Recommendation (<http://www.w3.org/TR/SVG/>) was published.

SVG, for which a lot of tutorial material and introductory Web sites exist, see, e.g. SVG Tutorial (<http://www.svgtutorial.com/>), is an XML vocabulary that provides an open-standard vector graphics language for two-dimensional graphics. It allows one to design Web pages with high-resolution graphics. SVG has three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and constructed using previously defined objects. Text can be in any XML namespace suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, gradients, animation, clipping paths, alpha masks, filter effects, template objects and extensibility.

SVG drawings can be dynamic and interactive. The DOM (Document Object Model) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as `onmouseover` and `onclick` can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page.

When native SVG support arrives in browsers Web pages will look a lot more like printed pages, with advanced typographic effects and pixel-perfect positioning of page elements. Pages will be custom-zoomable, and anti-aliasing will make every element look crisp at every magnification. Pages will also be faster to download, because text can be gzip-compressed, and some of the most elaborate art and animation effects will be vector art, which is usually much smaller than bitmapped images.

SVG is a CSS-compliant, Unicode-ready XML grammar for encoding text and/or graphics. That means you can embed SVG files inside HTML or XML, using `embed` or `object` elements, or you can use SVG as a standalone document type. Moreover, inside SVG files, you can embed inline graphics of the JPEG or PNG variety, using the `image` element, so that is not limited to stroked or filled vector drawing. Text or graphics can be styled with CSS, which can be embedded, called inline, or externally linked. This makes SVG a very flexible and powerful document standard.

One can use scripting languages, such as JavaScript or perl, or languages, such as java, to access and manipulate SVG page components at runtime via the DOM. This means that every style property (fill color, stroke width, opacity, etc.) for every art or text element on an SVG page can be manipulated using JavaScript. Every x-y position attribute for every element can

be manipulated; hence, objects can be moved, or animated. Even the raw transform matrices for objects (scaling, rotation, and shearing happen) can be tweaked on an object-by-object basis at runtime.

## 5.2 Generating SVG instances and T<sub>E</sub>X fonts

A large fraction of all scientific documents with rich content are marked up with L<sup>A</sup>T<sub>E</sub>X. On the other hand XML has become a *lingua franca* of the Internet and XML-aware tools have been ubiquitous. Therefore, it is important to optimally integrate L<sup>A</sup>T<sub>E</sub>X and XML, especially is one has to display graphics images that contain T<sub>E</sub>X fonts. In this section we describe work carried out in the framework of the TIPS Project to generate SVG instances of the most-used T<sub>E</sub>X font families, so that they can be rendered in SVG-capable browsers.

It must be realized that SVG provides only the final-form two-dimensional view on the output medium, so that paragraphs or pages must be formatted by the application upstream (e.g., T<sub>E</sub>X, drawing tools). It is possible to translate EPS files to SVG with Adobe Illustrator (commercial) or with Wolfgang Glunz' `pstoedit` (the SVG translator option is shareware). For direct translation from DVI there is Adrian Frischauf's `dvi2svg`, mentioned earlier. The translations work quite well as long as one uses standard fonts, such as Times, Helvetica and Courier. However, these applications have some problems with T<sub>E</sub>X fonts and their non-standard character encodings.

### 5.2.1 Choice of a font encoding

This problem of font encoding is closely related to the fact that SVG, being an XML language, uses Unicode as basic character encoding. So if one wants to go from T<sub>E</sub>X (DVI) to SVG the driver has to map all T<sub>E</sub>X characters to their Unicode code-point and use a *large* corresponding SVG font that encodes all the needed characters. Such an full mapping, although not impossible, is far from trivial. Hence, in coordination with Glunz, we have opted for a temporary hack, where we use a special *ad-hoc* option for `pstoedit`, where we map the T<sub>E</sub>X-code into Unicode's private user area using code positions hex U+E000 to U+E0FF for each font instance.

### 5.2.2 Generation of the SVG instances

Two procedures were used to generate SVG instances for T<sub>E</sub>Xs Computer Modern (CM) font family.

Working from the CM Metafont sources we used Szabó Péter's (`TeXtrace` (<http://www.inf.bme.hu/~pts/textrace/>)). `TeXtrace` is a collection of scripts for Unix that convert any T<sub>E</sub>X font into a Type1 `.pfb` outline font immediately suitable for use with `dvips`, `pdftex`, Adobe `acroread` and many other programs. It now also has an option to generate SVG but this did not directly what we needed and we had to add some code to make this work. We in fact preferred to use `TeXtrace` to generate `pfb` files, and fall back on the second approach, that we describe next.

Working from the Type 1 `pfa/pfb` font sources one can rather easily generate the corresponding SVG font. We developed a Perl script `type1SVG` to achieve this translation, where the only (minor) difficulty was the different behaviour concerning the current point between the `closepath` operators in Type1 and SVG, which added some bookkeeping complexity to the problem.

Comparing hand-crafted CM Type1 fonts with those generated by `TeXtrace`, we found that the former amounted to only half the size of the latter, which seems due to the fact that less control points are used. On the other hand, hints were not used for the SVG renderings. Nevertheless we feel that the two techniques (rendering with `TeXtrace` or going directly from hand-crafted Type1 fonts) are needed since in some cases for some publicly available Type1 fonts no MF fonts exist, while for some fonts only the Metafont sources are (publicly) available.

Note that SVG fonts are not needed if one is content with having the complete EPS figure, including the text, translated into SVG as pure paths descriptors (the text characters are treated merely as graphics paths). Use `pstoedit`'s `-dt` option in that case. This allows for fast rendering but loses the structural information about the graphics, making it more difficult to update.

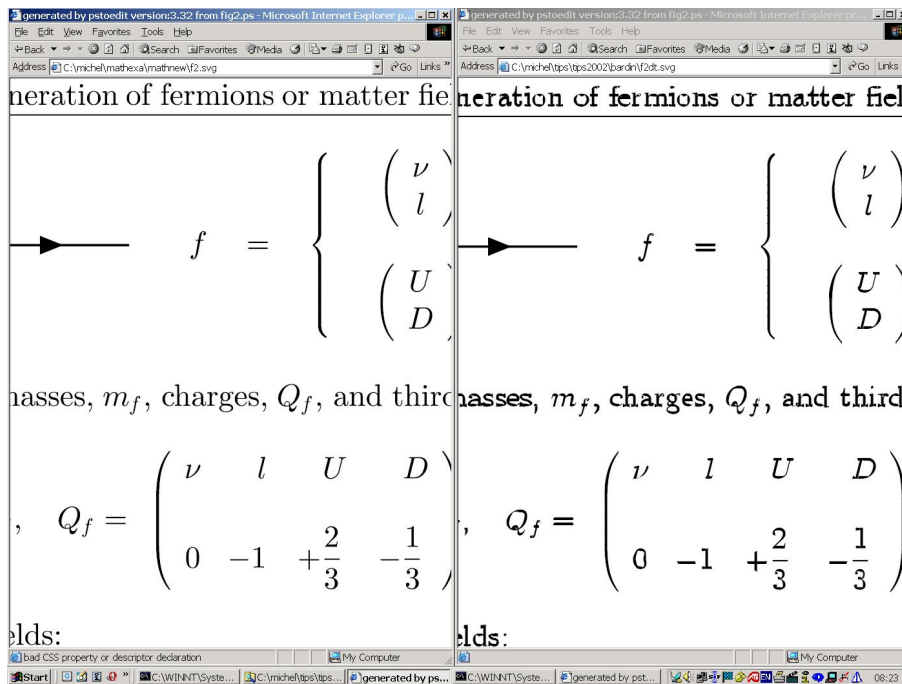


Figure 19: Text rendered with SVG fonts or as graphics paths

Moreover, when looking closer at the quality of the rendering it is clear that the  $\text{T}_{\text{E}}\text{X}$  SVG fonts display much better than considering the letters as SVG paths (right part of Figure 19).

The SVG document generated with `dvi2svg` from a DVI file will include the SVG outlines for the needed characters (font sub-setting is used), while for the SVG generated with `pstoedit` from PostScript the needed characters glyphs can be included with an XSLT stylesheet `inssvg.xsl`, which works with most commonly available XSLT processors. The SVG font set for the CM families is available as a ZIP file (<http://goossens.home.cern.ch/goossens/svgfonts.html>) and can also be obtained from the `dvi2svg` Website.

The SVG form of the test page for the complete CM symbol font `cmsy10` is shown in Figure 20, which displays the page as viewed with Microsoft Internet Explorer 6 equipped with Adobe's `svgview` plug-in.

Figure 21 shows a series of Feynman diagrams and their corresponding propagators. The page was prepared in  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and translated into SVG with `pstoedit`. We included the needed  $\text{T}_{\text{E}}\text{X}$  fonts into the SVG file and display the result with Apache Batik SVG browser.

## 6 A modular XML-based markup language for the sciences

Peter Murray-Rust and Henry S. Rzepab, when developing earlier versions of CML (Chemical Markup Language), realised that several components of the CML vocabulary were not specific to chemistry only (e.g. temperature, pressure, etc.) but could be used in other application areas. Therefore, they modularised CML 1.04 into two parts: purely chemical core components, e.g. the representation of molecules, and the general infrastructure for a modular XML-based markup language for use in Scientific, Technical and Medical publishing, STM-ML 1.0 (<http://www.xml-cml.org/stmml/>).

As the problem that they address is of a quite general and fundamental nature it is likely that the scientific community will develop several (possibly independent) approaches, although at present no such generally available specifications are known to exist. On the other hand, some markup languages for modeling specific domains have been developed, e.g.,

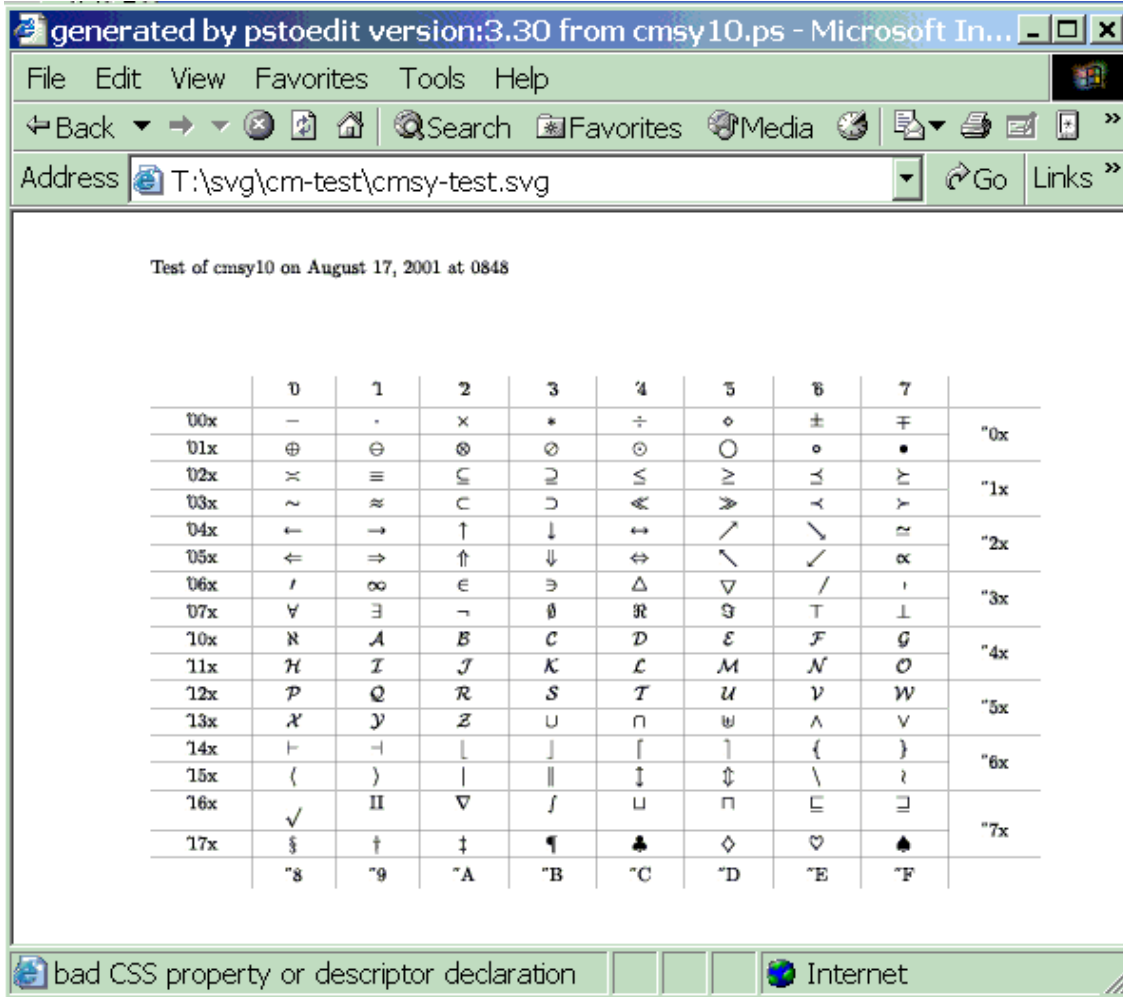


Figure 20: SVG version of the CMSY10 font table (viewed with Microsoft Internet Explorer 6).

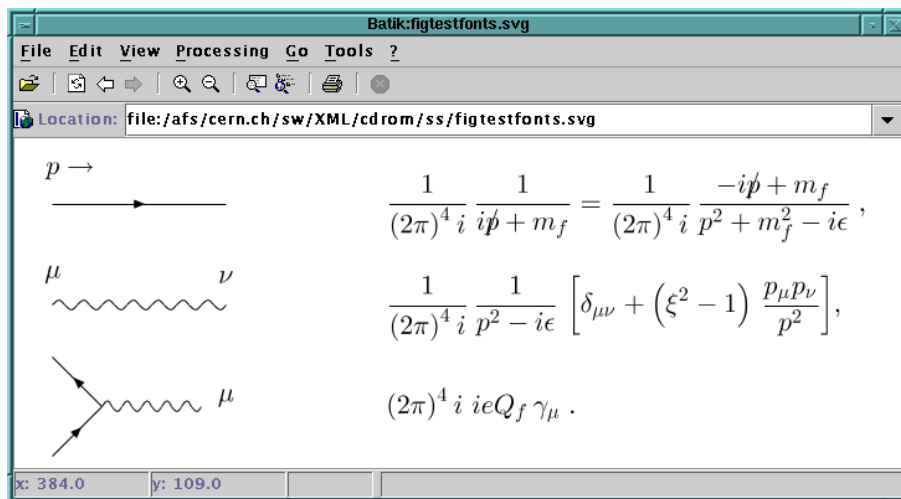


Figure 21: A Feynman diagrams and their propagators (SVG views).

- CellML (<http://www.cellml.org/>) who lets one store and exchange computer-based biological models. CellML allows scientists to share models even if they are using different model-building software. It also enables them to reuse components from one model in another, thus accelerating model building.
- SBML, the Systems Biology Markup Language (<http://www.cds.caltech.edu/erato/>), which represents biochemical network models, thus allowing simulator groups to exchange data saved in SBML format.
- FieldML (<http://www.physiome.org.nz/sites/physiome/fieldml/pages/>), for describing time-varying and spatially-varying coordinate information, and which is used by AnatML, the Anatomical Markup Language (<http://www.physiome.org.nz/sites/physiome/anatml/pages/index.html>), for storing geometric information and documentation obtained as part of the musculoskeletal modeling project.

All these languages cover some of the same concepts as STM-ML (e.g. units, metadata), but they represent information usually at a higher level of abstraction. Moreover, just like STM-ML, they use a modular approach and re-use other specifications (e.g. MathML, RDF) when needed rather than specifying their own. For instance, POSC (Petrotechnical Open Software Corporation) has done some work for developing a schema for units (<http://www.posc.org/ebiz/pefxml/patternsobjects.html>).

## 7 Recommendations and conclusion

Based on our investigation of up-translating several kinds of scientific documents we were able to come up with a few authoring guidelines for documents that have to be re-used on the Web.

Leave  $\LaTeX$  or other document processing commands as generic as possible, i.e., clearly separate content and presentation issues. Refrain from fine-tuning the visual placement of math, graphics, etc. in a given document. In particular explicit spacing commands are only valid in a given context, and will end up, e.g., in the MathML and result in funny and mostly unwanted results on other output media.

Provide explicit code for alternate resources. For instance, for a math formula one could have the original  $\TeX$  code, the MathML translation, the SVG graphical representation and a PNG image. This will ensure an optimal treatment on all output media.

Work is going on in many technical, scientific, and medical areas to define XML-based languages that capture the model content of their respective domains. In several cases (units, metadata, graphics, math) the same problems are faced, so that developments by all these groups should be re-used as far as possible, and collaborative efforts set up to the benefit of all communities.

It is always advisable to supplement semantic content that has a graphical representation with an SVG (and possibly a PNG bitmap) rendering. Of course the CML-encoded chemical formulae, or other ad-hoc formats, such as MDL MOL (<http://www.mdli.com/>) and PDB (<http://www.rcsb.org/pdb/>) files, etc. could be added as specific formats in the given application domain for optimal re-use.

XML-aware browsers, which are becoming more readily available, can now fully exploit the information present in XML documents and plug-ins or native support for some vocabularies (e.g., SVG, MathML, CML) is starting to appear. Thus it becomes advantageous to translate scientific documents marked up in specific languages, such as  $\LaTeX$ , into XML. A lot of work still remains to be done, and it is the user community that will have to push the browser developers into ever better support of the technologies they need.

Figure 22 shows various formats in which electronic scientific documents can be prepared for the Web. It also shows how to transform between various formats. The XML document source marked up semantically using one more XML vocabularies is represented by the large rectangle at the right. One can display these sources directly using stylesheets on XML-enabled browsers, or one can transform the XML into HTML first, transforming non-textual images into bitmaps for

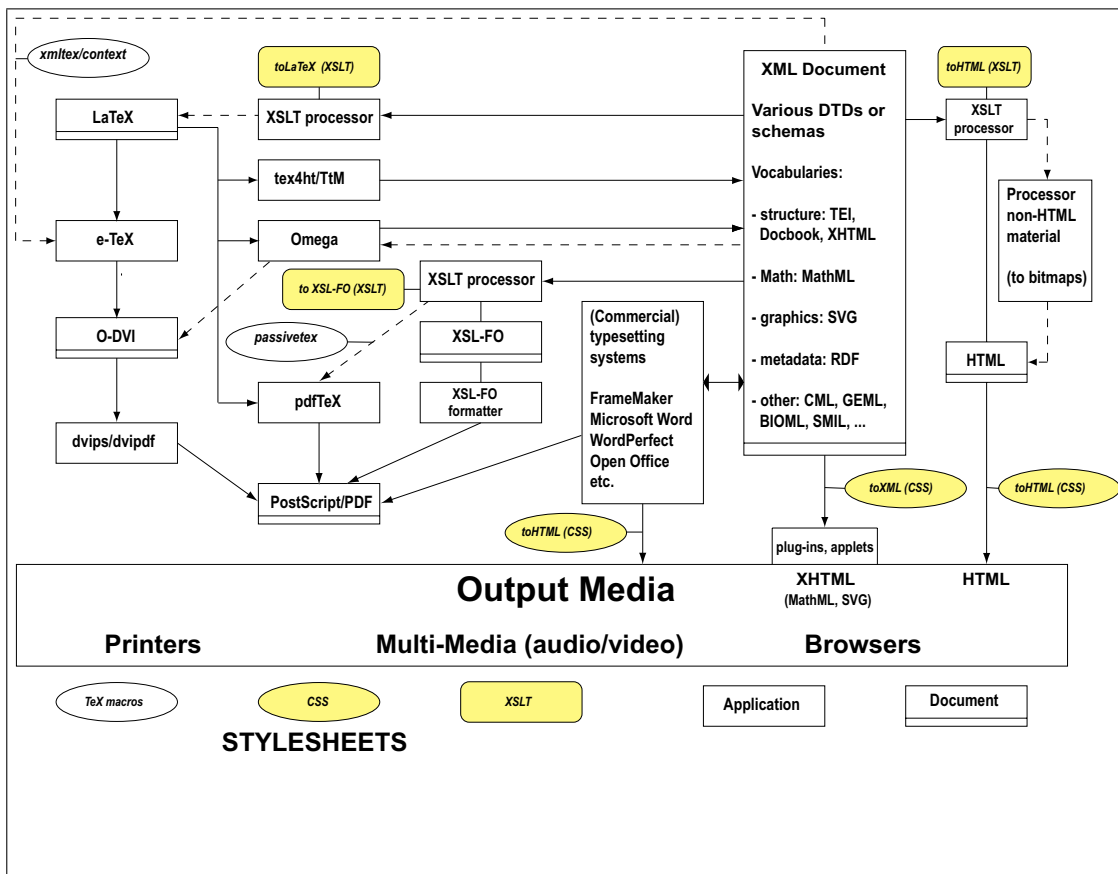


Figure 22: Electronic documents and the Web

easier viewing. At the left side of the figure we see  $\text{\TeX}$ 's formatting machinery, frequently used in the sciences. It can generate PostScript or PDF, that allows for typographically excellent output on high definition printing devices. Various pathways between  $\text{\LaTeX}$  and XML are shown. The link with other commercial and free typesetting packages, such as Adobe FrameMaker, Microsoft Word, or Corel WordPerfect is also possible using XML transformations.

It is important to realise that scientists do not need to master all the details of the XML language. They can continue to use the authoring tools that they are most comfortable with. Applications will deal with the translation of the ad-hoc formats into XML for higher portability. The relation between the various formats can be expressed in an unambiguous way by a standard resource description language, such as RDF (<http://www.w3.org/RDF/>), and a vocabulary, such as a Dublin Core (<http://dublincore.org/>). There one can specify such things like author, title, abstract, set of keywords, availability, copyright, etc. This allows applications, such as data mining or database services, Internet searches, that need to partially re-use material, to be optimally informed about the content of the documents.

In summary, every branch of science can choose how to optimally represent the information to be stored in a semantic way using a language (or languages) specifically suited for modeling knowledge in the given application domain. Then, for communicating this knowledge, its semantic content should be translated into a presentation form, also called *final form*, for the chosen output medium. To these final form languages, such as SVG (graphics), XSL-FO (linear paged output), Speech Markup, one can add animation or interactive control with programming languages, such as Java, or scripting languages, such as Javascript, perl, or python. This final form transformation, however, is lossy and one-way, with the material exclusively suitable for viewing, hearing, etc.